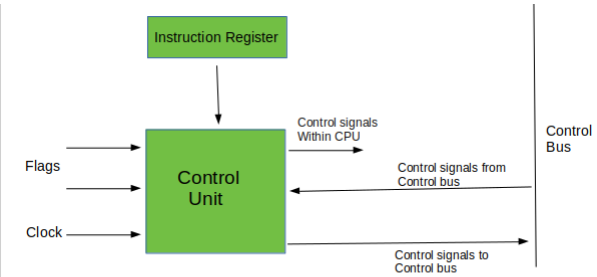
Unit-IV

Control Unit and its Design

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.
A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. Examples of devices that require a CU are:

- Control Processing Units(CPUs)
- Graphics Processing Units(GPUs)



Block Diagram of the Control Unit

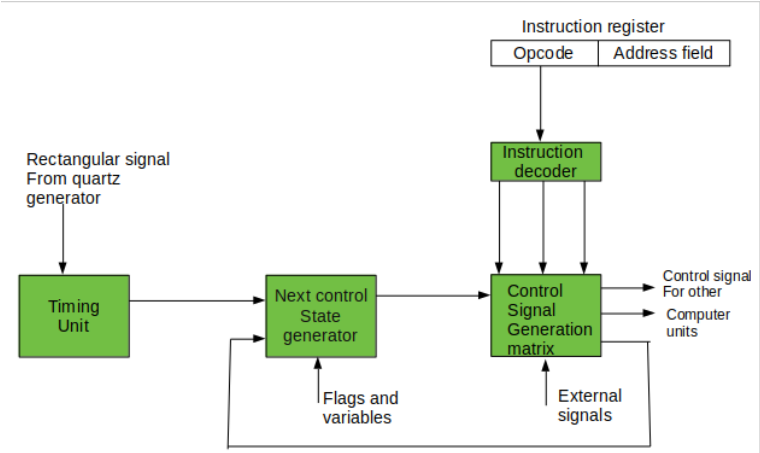
Functions of the Control Unit –

1. It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
2. It interprets instructions.
3. It controls data flow inside the processor.
4. It receives external instructions or commands to which it converts to sequence of control signals.
5. It controls many execution units (i.e. ALU, data buffers and registers) contained within a CPU.
6. It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

Types of Control Unit –
There are two types of control units: Hardwired control unit and Microprogrammable control unit.

1. **Hardwired Control Unit –**
   In the Hardwired control unit, the control signals that are important for instruction

execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode.
As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for executive units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.



Block diagram of a hardwired control unit of a computer

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit.

A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state.

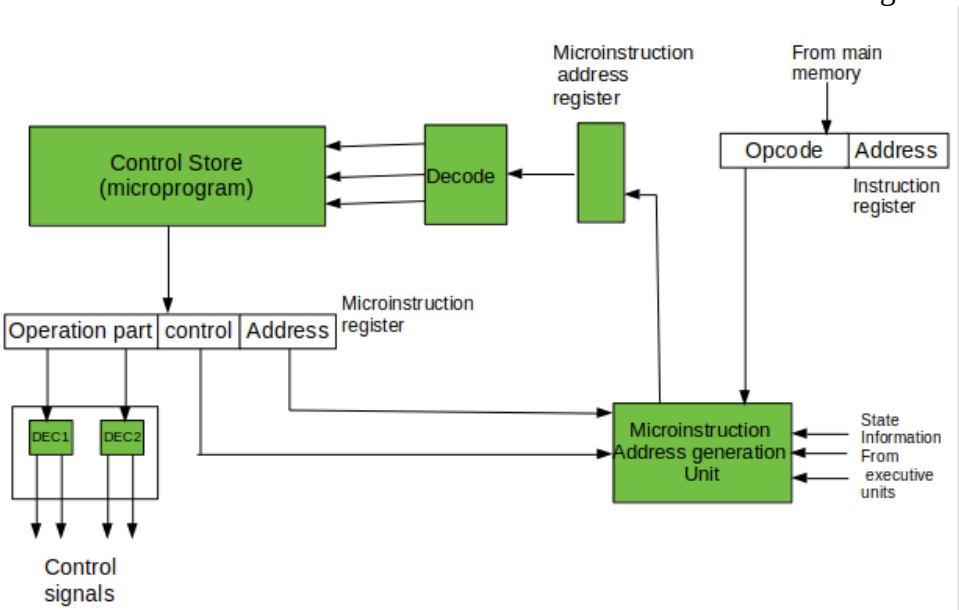
This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). zThe values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle.

The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer

register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

2. Microprogrammable control unit –
The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution.
In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

With a single-level control store:
In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a microprogram that interprets execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoders, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction microprogram and a control field used to control activities of the microinstruction address generator.

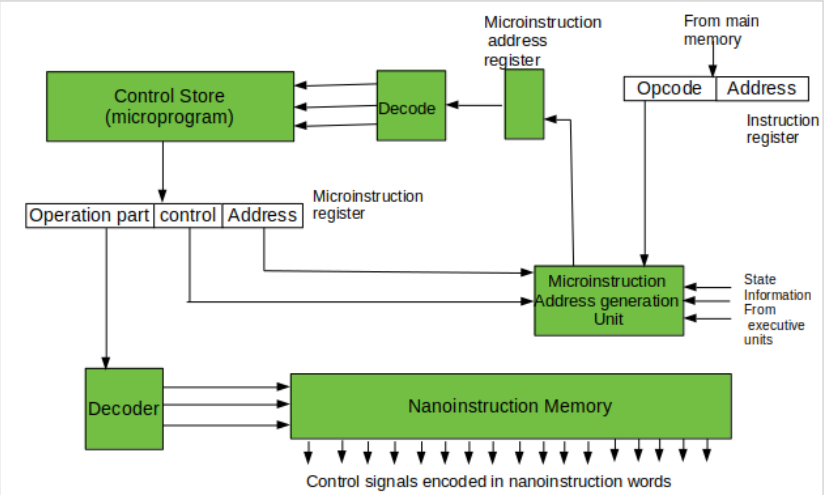


Microprogrammed control unit with a single level control store

The last mentioned field decides the addressing mode (addressing operation) to be applied to the address embedded in the ongoing microinstruction. In microinstructions along with conditional addressing mode, this address is refined by using the processor condition flags that represent the status of computations in the current program. The last microinstruction in the instruction of the given microprogram is the microinstruction that fetches the next instruction from the main memory to the instruction register.

With a two-level control store:
In this, in a control unit with a two-level control store, besides the control memory for microinstructions, a nano-instruction memory is included. In such a control unit, microinstructions do not contain encoded control signals. The operation part of microinstructions contains the address of the word in the nano-instruction memory, which

contains encoded control signals. The nano-instruction memory contains all combinations of control signals that appear in microprograms that interpret the complete instruction set of a given computer, written once in the form of nano-instructions.



Microprogrammed control unit with a two-level control store

In this way, unnecessary storing of the same operation parts of microinstructions is avoided. In this case, microinstruction word can be much shorter than with the single level control store. It gives a much smaller size in bits of the microinstruction memory and, as a result, a much smaller size of the entire control memory. The microinstruction memory contains the control for selection of consecutive microinstructions, while those control signals are generated at the basis of nano-instructions. In nano-instructions, control signals are frequently encoded using 1 bit/ 1 signal method that eliminates decoding.

Computer Instructions Formats

Computer instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.

An instruction comprises of groups called fields. These fields include:

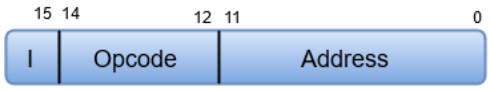
- The Operation code (Opcode) field which specifies the operation to be performed.
- The Address field which contains the location of the operand, i.e., register or memory location.
- The Mode field which specifies how the operand will be located.

A basic computer has three instruction code formats which are:

1. Memory - reference instruction
2. Register - reference instruction

3. Input-Output instruction

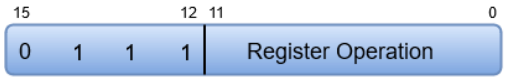
Memory - reference instruction



(Opcode = 000 through 110)

In Memory-reference instruction, 12 bits of memory is used to specify an address and one bit to specify the addressing mode 'I'.

Register - reference instruction



(Opcode = 111, I = 0)

The Register-reference instructions are represented by the Opcode 111 with a 0 in the leftmost bit (bit 15) of the instruction. A Register-reference instruction specifies an operation on or a test of the AC (Accumulator) register.

Input-Output instruction

(Opcode = 111, I = 1)

Just like the Register-reference instruction, an Input-Output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of the input-output operation or test performed.

Note

- The three operation code bits in positions 12 through 14 should be equal to 111. Otherwise, the instruction is a memory-reference type, and the bit in position 15 is taken as the addressing mode I.
- When the three operation code bits are equal to 111, control unit inspects the bit in position 15. If the bit is 0, the instruction is a register-reference type. Otherwise, the instruction is an input-output type having bit 1 at position 15.

Instruction Cycle

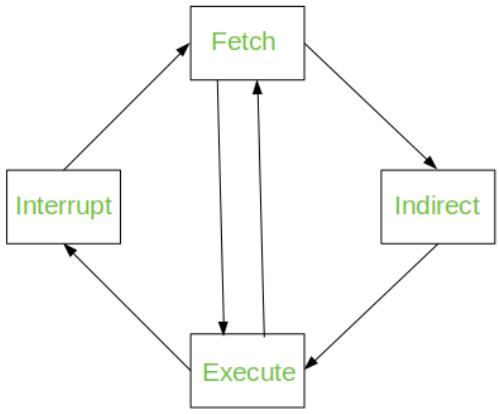
Registers Involved In Each Instruction Cycle:

- **Memory address registers (MAR)**: It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.

- **Memory Buffer Register (MBR)**: It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
- **Program Counter(PC)** : Holds the address of the next instruction to be fetched.
- **Instruction Register(IR)** : Holds the last instruction fetched.

The Instruction Cycle –

Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations. In the above examples, there is one sequence each for the *Fetch*, ***Indirect, Execute and Interrupt Cycles.***



The Instruction Cycle

The *Indirect Cycle* is always followed by the *Execute Cycle*. The *Interrupt Cycle* is always followed by the *Fetch Cycle*. For both fetch and execute cycles, the next cycle depends on the state of the system we assumed a new 2-bit register called *Instruction Cycle Code* (ICC). The ICC designates the state of processor in terms of which portion of the cycle it is in:-

00	:	Fetch	Cycle
01	:	Indirect	Cycle
10	:	Execute	Cycle

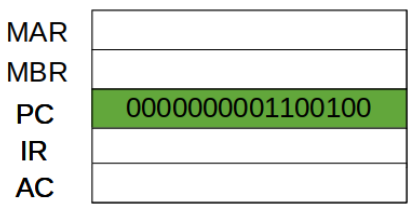
11 : Interrupt Cycle

At the end of the each cycles, the ICC is set appropriately.The above flowchart of *Instruction Cycle* describes the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern (this is a simplified example). The operation of the processor is described as the performance of a sequence of micro-operation.

Different Instruction Cycles:

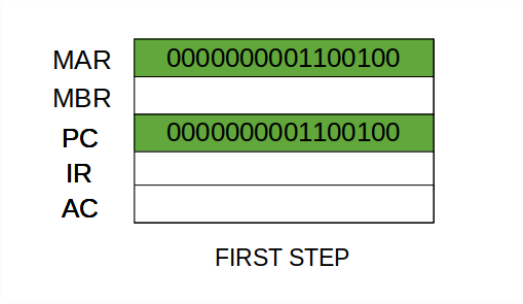
1. **The Fetch Cycle –**

At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC).

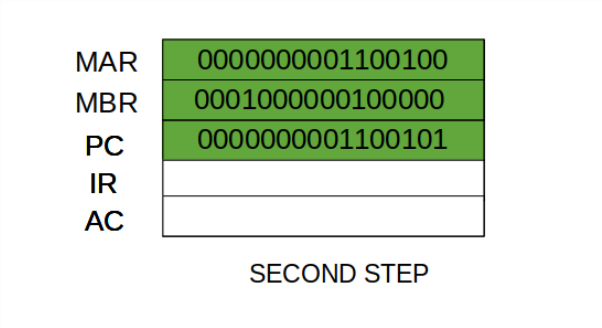


BEGINNING

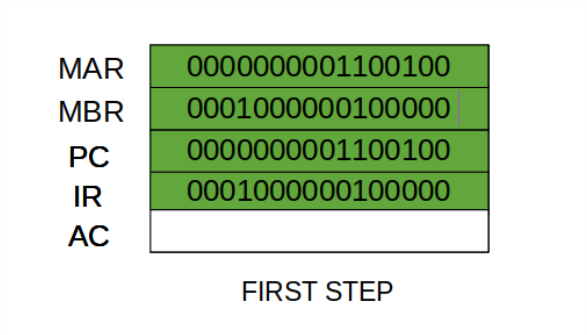
Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.



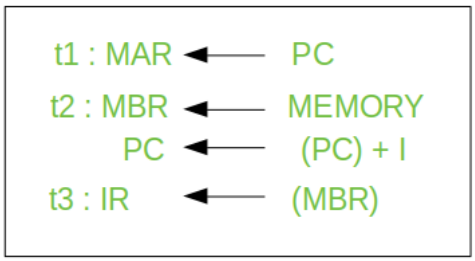
Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction.(These two action can be performed simultaneously to save time)



Step 3: The content of the MBR is moved to the instruction register(IR).



Thus, a simple *Fetch Cycle* consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:-



Here 'I' is the instruction length. The notations (t1, t2, t3) represents successive time units. We assume that a clock is available for timing purposes and it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit.
First time unit: Move the contents of the PC to MAR.
Second time unit: Move contents of memory location specified by MAR to MBR. Increment content of PC by I.
Third time unit: Move contents of MBR to IR.
Note: Second and third micro-operations both take place during the second time unit.

2. **The Indirect Cycles –**
Once an instruction is fetched, the next step is to fetch source operands. *Source Operand* is being fetched by indirect addressing (it can be fetched by any addressing mode, here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following *micro-operations* takes place:-

Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.
Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)
Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

Note: Now IR is ready for the execute cycle, but it skips that cycle for a moment to consider the *Interrupt Cycle* .

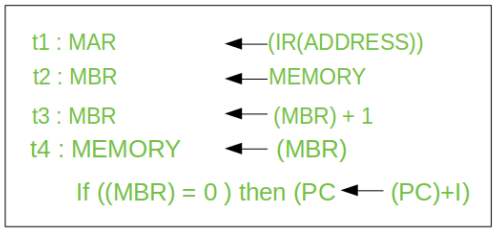
3. **The Execute Cycle**
The other three cycles(*Fetch, Indirect and Interrupt*) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.
Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.
Lets take an hypothetical example :-
consider an add instruction:

Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

We begin with the IR containing the ADD instruction.
Step 1: The address portion of IR is loaded into the MAR.
Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.
Step 3: Now, the contents of R and MBR are added by the ALU.

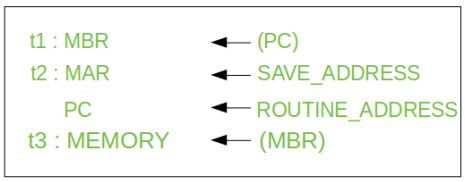
Lets take a complex example :-

Here, the content of location X is incremented by 1. If the result is 0, the next instruction will be skipped. Corresponding sequence of micro-operation will be :-



Here, the PC is incremented if (MBR) = 0. This test (is MBR equal to zero or not) and action (PC is incremented by 1) can be implemented as one micro-operation.
Note : This test and action micro-operation can be performed during the same time unit during which the updated value MBR is stored back to memory.

4. **The Interrupt Cycle**:
At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The nature of this cycle varies greatly from one machine to another.
Lets take a sequence of micro-operation:-



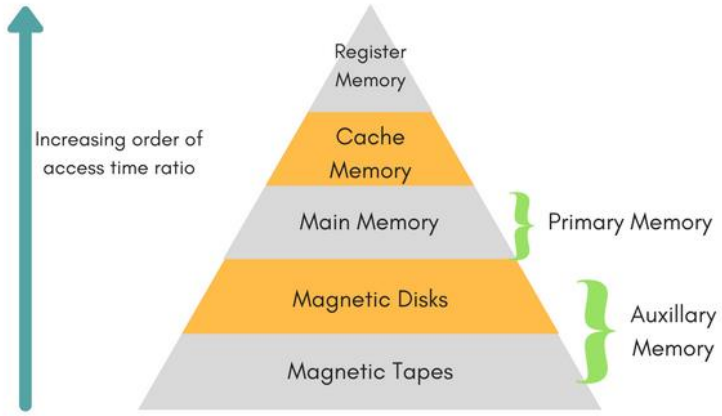
Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.
Step 2: MAR is loaded with the address at which the contents of the PC are to be saved. PC is loaded with the address of the start of the interrupt-processing routine.
Step 3: MBR, containing the old value of PC, is stored in memory.

Memory Concept

A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Generally, memory/storage is classified into 2 categories:

- **Volatile Memory**: This loses its data, when power is switched off.
- **Non-Volatile Memory**: This is a permanent storage and does not lose any data when power is switched off.

Memory Hierarchy



The total memory capacity of a computer can be visualized by hierarchy of components. The memory hierarchy system consists of all storage devices contained in a computer system from the slow Auxiliary Memory to fast Main Memory and to smaller Cache memory.

Auxillary memory access time is generally **1000 times** that of the main memory, hence it is at the bottom of the hierarchy.

The **main memory** occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O).

When the program not residing in main memory is needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space in main memory for other programs that are currently in use.

Input/Output Subsystem

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Peripheral Devices

Input or output devices that are connected to computer are called **peripheral devices**. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called **peripherals**.

For example: *Keyboards*, *display units* and *printers* are common peripheral devices.

There are three types of peripherals:

1. **Input peripherals** : Allows user input, from the outside world to the computer. Example: Keyboard, Mouse etc.

2. **Output peripherals**: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc

3. **Input-Output peripherals**: Allows both input(from outside world to computer) as well as, output(from computer to the outside world). Example: Touch screen etc.

Interfaces

Interface is a shared boundary btween two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

There are two types of interface:

1. CPU Inteface

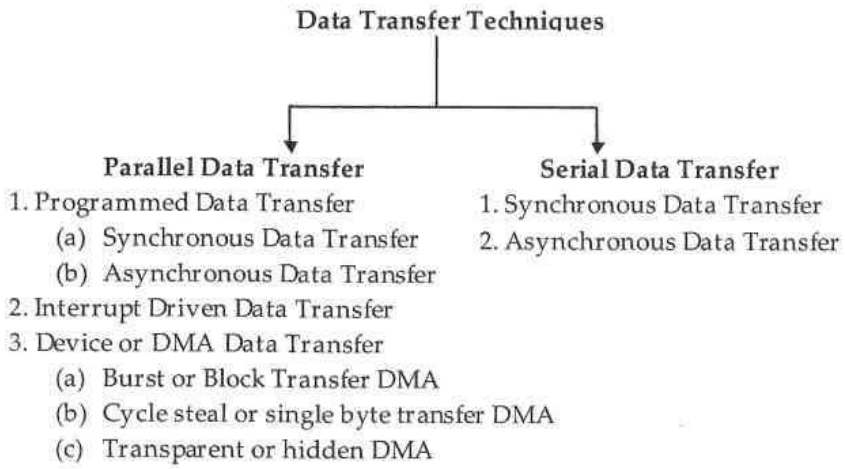
2. I/O Interface

Let's understand the I/O Interface in details,

Input-Output Interface
Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called **input-output interface units** because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

Modes of I/O Data Transfer

We can broadly classify the data transfer schemes into two modes – Serial Data Transfer and Parallel Data Transfer.



Classification of the data transfer techniques in 8085

Our device, the Intel 8085 Microprocessor, is a parallel device. Thus, it transfers 8 bits of information simultaneously over 8 data lines in the parallel I/O mode. But sometimes, there are instances where using this data transfer technique might be just theoretical or impossible to apply. Using parallel data transfer can be expensive if communication is to take place over vast distances. Also, if the device, on the other hand, follows the protocol of serial communication only, then it is impossible to use parallel mode.

Difference between Parallel Data Transfer and Serial Data Transfer

As discussed earlier, we can broadly classify data transfer schemes into parallel data transfer techniques and serial data transfer techniques. Let us understand the differences between them.

SERIAL DATA TRANSFER TECHNIQUES	PARALLEL DATA TRANSFER TECHNIQUES
Under this scheme, the data is transferred one bit at a time.	Under this scheme, the data is transferred several bits at the same time.
It is a slower mode of data transfer.	Data is transferred much quicker.
Serial data transfer is preferred when data is to be sent over a long distance and the cost of cables would be too expensive.	Parallel data transfer is the preferred technique for short-distance communication.
For this mode, the transmitter first performs parallel – to – serial conversion and the serial – to – parallel conversion at the receiver.	No such conversions are required at both the transmission and reception endpoints.
This mode requires a single line to transfer information.	This mode requires multiple lines for data transfer.
Noise and errors are much lesser.	As multiple bits are transmitted at the same time, there is scope for more error and noise.

Cables used for serial communication are much thinner, longer, and very economical.	Here, the cables are much shorter, and thicker compared to the Serial communication cables.
This is a very reliable, inexpensive, and straightforward process.	It is considered a little unreliable, expensive, and complicated process.

Parallel Data Transfer Techniques

We know that under the parallel data transfer scheme, multiple data bits can be transmitted at the same time. Thus, for the Intel 8085, 8 bits of data are sent all together using eight parallel lines. Let us go through the different types of parallel data transfer schemes. We have:

- Programmed I/O Data Transfer
- Interrupt Driven I/O Data Transfer
- Direct Memory Access (DMA) Data Transfer

Let us study each of these transfer schemes in detail.

Programmed I/O Data Transfer

- This is a straightforward scheme under parallel data transfer mechanisms. This mode is generally preferred for simple, small microprocessor systems where speed is critical.
- This method can work under the synchronous and asynchronous mode, depending on the speed and architecture of the I/O devices.
- We also prefer this method when there is a small amount of information to be exchanged between the microprocessor and other devices that are placed near to the microprocessor. Example: Computer, printer, etc.
- Using the IN and OUT instructions, data transfer is carried out between the microprocessor and I/O devices.
- The processor reads the data from an input port or input device using the IN command. The processor sends data out from the CPU to the output port or the output device using the OUT instruction.
- Thus, as the speeds of the processor and external device match, the data transferring process is carried out using the IN and OUT instructions.

Synchronous Data Transfer Method

- The word '**Synchronous**'means 'taking place at the same time.'
- Thus, to establish communication between our processor and the device, we need to set a common clock pulse. This common pulse ***synchronizes*** the peripheral device with the 8085 microprocessor.
- This method is used when the speed of the microprocessor, Intel 8085, in this case, and the external peripheral device match with each other.
- If the device is ready to send data, it can indicate via the **READY** pin of 8085. Once the speeds match, the data transfer immediately begins, once a signal is issued by the microprocessor to begin transferring. The microprocessor need not wait for an extended period because of the matching speeds.
- This technique of data transfer is seldom used to communicate with I/O devices though. Because I/O devices compatible with the microprocessor's speed are usually not found.

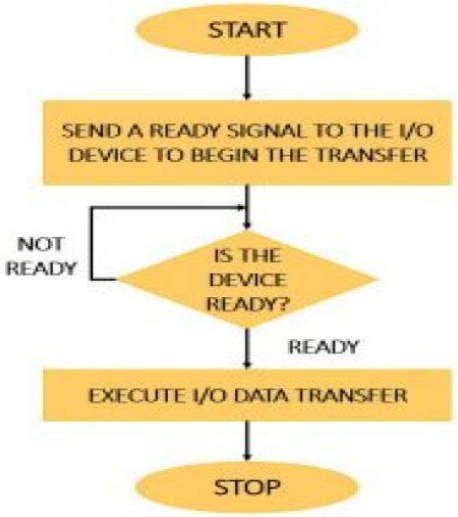
- Hence, this method of data transfer is most commonly employed for communicating with compatible memory devices.

Asynchronous Data Transfer Method

- When the speed of the I/O device is slower than that of the microprocessor, we prefer the Asynchronous Data Transfer Method. As the speeds of both the devices differ, the I/O device's internal timing is entirely independent of the microprocessor.
- Thus, they are termed to be '***asynchronous***' from each other. The term asynchronous means 'at irregular intervals.'

We implement the Asynchronous Data Transfer Method using the ***handshaking*** policy. But how is this method applied? And what happens next?

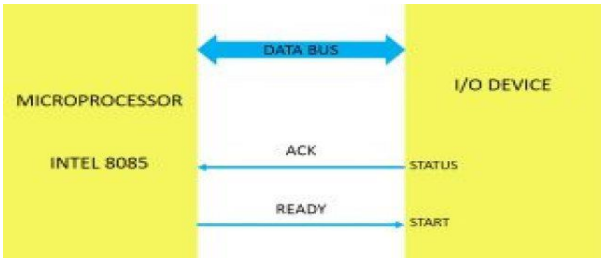
Under the handshaking method, the microprocessor and I/O device exchange a few signals before beginning the transfer of data between them.



let us understand the flowchart of the handshake protocol.

- First, the microprocessor raises a ready signal and sends it to the I/O device, which is connected to it.
- The status of the I/O device is continually checked to see it is prepared for data transferring.
- If the device is not ready, it is checked again and again until the device signals it is ready.
- Once the device is ready, the data transfer begins from the microprocessor to the I/O device.

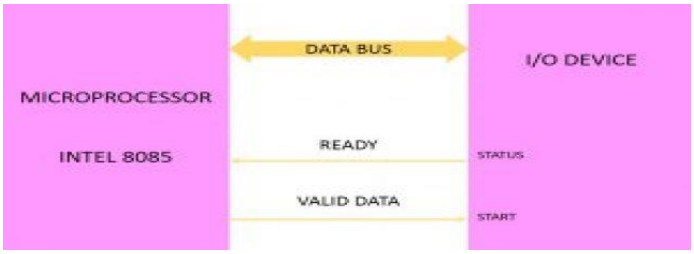
But how does the I/O device inform the microprocessor that it is ready for data transfer? This is done by the 'ACK' (**Acknowledge**) Signal. Also known as the handshake signal.



The microprocessor first checks the readiness of the I/O device continually. Once the I/O device is ready to accept data from the microprocessor, it sends an ACK signal to the microprocessor. This indicates the microprocessor that the device is all set for the reception of data. Now the data transmission can finally take place.

What happens if the opposite is supposed to happen? What if the I/O device has to submit information to the microprocessor?

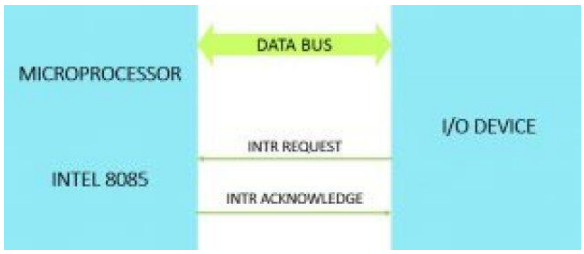
In such a situation, the I/O device issues the ready signal back to 8085, informing it that it is ready to send data to the processor. In response to this ready signal, a valid data signal is sent by the 8085 to the I/O device, and then the valid information is put on the common data bus for the data transfer.



We can conclude by saying that under the Programmed I/O Data Transfer method, the microprocessor is always busy checking the status of the slower I/O device continuously for sharing data. Thus, some amount of time is wasted on the microprocessor's behalf. Additionally, this method of data transfer is used by I/O devices and slow external memory peripherals.

Interrupt Driven I/O Data Transfer

The Programmed I/O Data Transfer, unfortunately, wastes a lot of time. Thus, the Interrupt Driven I/O Data Transfer is a much better option. Here, no extra time of the microprocessor is wasted in waiting for a response/signal from the external device. Under this method, the I/O device informs the microprocessor about its readiness, only when it is ready. This is accomplished by interrupting the Intel 8085.



- To begin with, the microprocessor initiates the transfer of data by sending a request to the external device to be 'get ready.'
- Following this, the processor continues working normally, executing the original program rather than wasting its cycles by keeping a check on the status of the external I/O device.

- Once the I/O device is all set to transfer information, it sends a control signal to the 8085 to inform its readiness. This control signal is called the Interrupt (INTR) signal.
- Once the INTR signal is received by 8085, it responds to it by submitting an INTR acknowledgment signal back to the external I/O device.
- When the I/O device gets the acknowledgment, both the devices are now prepared for the data transfer. The microprocessor completes the execution of the current instruction, then suspends the job. It saves all the contents and the status of the PC (Program Counter) into the stack memory and then proceeds with the subroutine program.
- This subroutine program for the raised interrupt is called the Interrupt Service Subroutine (ISR) program.
- This ISR first saves the status of our processor into the stack.
- Once the data transfer is completed by executing the necessary instructions, the ISR restores the previous processor status and continues with the normal execution of the main program.

Direct Memory Access (DMA) Data Transfer

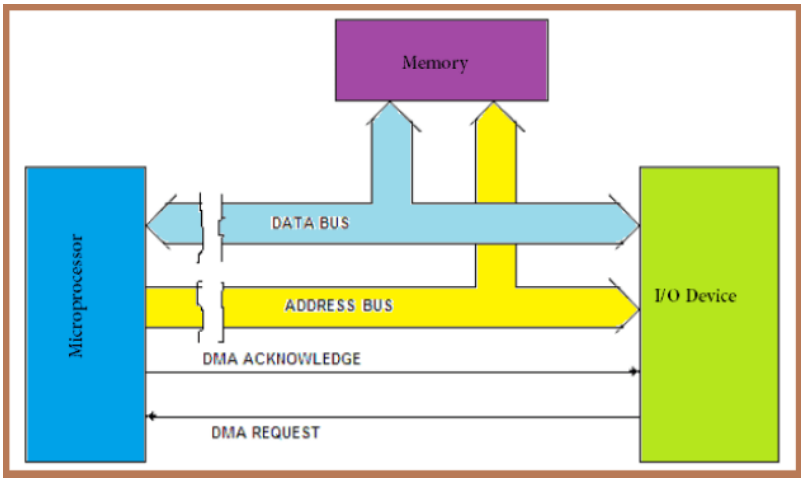
In the two schemes discussed earlier, the transfer of information between the I/O device and the mass storage device/memory is via the Accumulator register. But if there is a bulk amount of data to be transferred between the I/O device and the microprocessor, then these methods are unworthy of the investment being put into them, and even time – consuming.

To overcome such circumstances, the Direct Memory Access Data Transfer is put into use. This is an ideal method used for data transfer of a considerable amount between the microprocessor and the I/O device. It is also considered the fastest data transfer scheme.

But what makes it different from the previous two methods?

In the DMA scheme, the data is transferred between the I/O device (or DMA controller, a special IC) and the external memory directly *without having any interference by the accumulator register*. The processor gives up its control over the address bus as well as the data bus to the I/O device or DMA controller, thus aiding the exchange of information between the source and the destination directly.

We will now understand the working principle of the Direct Memory Access Data Transfer method.



Direct Memory Access Data Transfer

- First of all, we need to inform the microprocessor before beginning the process. For this, an I/O device first sends a request to a DMA controller using the DMARQ signal, which in turn forwards it to the processor in the form of a HOLD signal.
- Once such a request is received by the microprocessor, it ceases its control over the address bus and data bus. It then informs the DMA Controller of the situation by sending an acknowledgment signal using the HLDA command.
- The controller then informs the external peripheral by sending a DMACK signal.
- The DMA controller then gains control over the two buses and monitors the transfer of information between the two locations (Memory and I/O).
- Once the entire data transfer between the I/O device and the external memory gets over, the DMA Controller withdraws its request for using the data and address bus by disabling the HOLD and DMACK signals, and thus the microprocessor gains control over them again.

There are basically two types of techniques under the Direct Memory Access Data Transfer:

1. Burst or block transfer DMA
2. Cycle steal or the single-byte transfer DMA

Burst or Block Transfer DMA

- This is the fastest DMA mode of data transfer.
- In the Burst mode, at least two or more bytes of data are transferred continuously i.e., the entire block of data is transferred in a straight sequence. Here, the microprocessor is disconnected from the main system during the data transfer, and thus, the processor is unable to execute any program on its own during the information transfer.
- In fact, in this mode, the DMA controller acts as a Master.
- If there are about N number of bytes to be transferred, then N number of machine cycles will be adopted into the working of the processor.
- The DMA controller first sends a HOLD signal to the microprocessor to request access for the system's buses, and in turn, wait for the HLDA signal.
- One the HLDA signal is received, the DMA controller gets access over the system bus and sends one byte of information.
- Once a single byte is sent, the memory address is incremented, the counter is decremented, and then the next byte is sent.
- Thus, following this technique, all data bytes are transferred between memory and I/O devices. Once all the information is sent, the DMA controller disables the HOLD signal.
- It then enters into the slave mode.
- This method is generally used for loading data files or important programs into the memory. However, it keeps the CPU inactive for relatively long periods.

Cycle steal or Single-Byte transfer DMA

- In this mode, only a single byte is transferred at a time.
- This is thus much slower than burst DMA.
- In the cycle-steal DMA, The DMA controller sends a HOLD signal to the microprocessor.

- It then waits for the HLDA signal in return.
- Once the HLDA signal is received, it gets access over the system buses and **executes one DMA cycle only**.
- After this transfer, the HOLD signal is disabled, and it enters into the slave mode.
- The processor thus gets back its control over the address and data bus and continues executing the following machine cycle.
- However, if the counter has not touched down to zero, and there is still data to be exchanged, then the DMA controller sends a HOLD signal again to the processor, and sends the next byte of the information block.

Thus, only one DMA cycle takes place between every two machine cycles of the processor, and the execution speed of the instructions in the microprocessor falls back a bit.

The DMA Controller obtains the access for the system buses by repeatedly issuing the requests using the *Bus Request (BR)* and *Bus Grant (BG)* signals until the entire data block has been transferred.

Though the information bytes are not transferred as fast as in the Burst mode, the only advantage of the cycle-steal mode is that the CPU does not remain idle for long durations of time, as in the burst DMA mode. This method is mainly used in controllers which are used in monitoring data in real-time.